Real-time Attention-based Conversational Agent

Abhishek Vaid
Department of Computer Science
San Jose State University
San Jose, USA
abhishek.vaid@sjsu.edu

William B. Andreopoulos
Department of Computer Science
San Jose State University
San Jose, USA
william.andreopoulos@sjsu.edu

Abstract—**Neural Machine Translation (NMT) is a prominent natural language processing technique that is being used to develop conversational AI technology. Traditional industrial chatbots often rely on scripted responses and lack the ability to provide real-time data-driven interactions. Developing advanced AI chatbots like Google Bard or ChatGPT are out of reach for smaller or medium-sized organizations due to their scale and associated costs. Chatbots are majorly restricted by the data on which they were trained on and have no knowledge of current events. This research project intends to research and develop an approach that enables chatbots to provide live and up-to-date information in their responses and can be developed in minimalistic costs so that even smaller or medium-level organizations can afford to provide interactive AI chatbots. We experiment with various techniques in terms of the type of data being used to harness live capabilities. We focus on optimizing the hyperparameters required for building a conversational AI agent and leverage open-source technologies to minimize costs. To ensure flexibility and affordability, we adopt a microservice architecture that combines Attention based NMT models and Transformer Models with live API services features, leveraging the RASA actions API. This approach allows us to develop a prototype of an advanced chatbot that goes beyond the traditional scripted responses by providing real-time information to users and being affordable to develop.**

Keywords— Neural Machine Translation, Conversational AI.

I. INTRODUCTION

A. Introduction to Conversational AI chatbots

The need for communication has played a vital role in the development of the human civilization. Most digital technology we enjoy today is at a basic level sharing and consumption of data in different forms. As computer science progresses, we see a shift in technology starting to surpass human intelligence not only in terms of speed but in terms of creation of ideas itself. Conversational AI chatbots are the future of automated communication. Different industries such as Medical, Industrial, Technical or service-based, nowadays use a chatbot to handle basic client requests. Most of these chatbots are pre-scripted and provide only a number of options to choose from for having a conversation. This discrepancy makes the communication experience very mechanical, as human beings by nature are emotional creatures and require motivation from a peer. Something that a chatbot cannot provide yet because of a structured set of replies and lack of a human real-world experience. For example, a regular chatbot if asked a question on cricket may give a generic reply of it being a sport and some facts, but our conversational AI may reply like a viewer and even give a specific point of view on different play styles and cricketers. Our research focuses on developing a Conversational AI chatbot that acts like a human in giving replies and can handle multiple questions from different industries and provide its view like a human being. The chatbot focuses on giving a more human experience to the user and become a next stepping stone in the advancement of Conversational AI technology.

B. History of Neural Machine Translation

Machine translation has been in development for a while. It started in 1933 when Soviet scientist Peter Troyanskii developed a machine to select and print a word from cards when one is translating from one language to another.

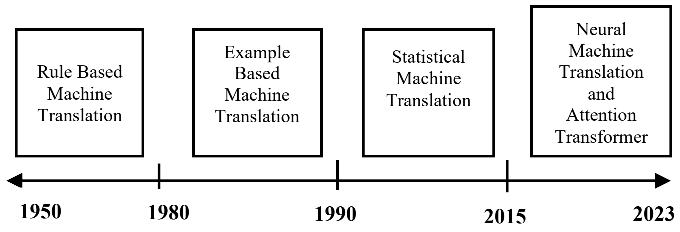

Rule Based Machine Translation
Example Based Machine Translation
Statistical Machine Translation
Neural Machine Translation and Attention Transformer
1950
1980
1990
2015
2023

Fig 1 Timeline for Machine Translation models

Then during the cold war in 1954 at IBM headquarters in New York a project called Georgetown-IBM experiment started for Russian to English translation, where in IBM developed a typewriter machine that translated Russian to English sentences. Subsequently, during the 70s Rule-based machine translation (RBMT) [1] came along. It had 3 ways to translate one language to another – Direct Machine translation, Transfer Based Machine Translation, and Interlingual Machine Translation. This lasted for a while but was not good enough. Then in 1984, a Japanese scientist named Makoto Nagao from Kyoto university proposed an idea that instead of converting each word, one at a time we can use examples of pre converted sentences and used them by swapping extra words to make a converted sentence from one language to another, this technique came to be known as Example-based Machine Translation (EBMT) [1]. This was a big milestone for Natural Language understanding as scientists for the first time understood that statistics could be used for language understanding prediction and conversion. In 5 years in 1990 at the IBM research center, IBM showcased the language machine translation system that converted one language to another by statically understanding word usage using millions of lines of conversation and splitting words to check their usage for types of sentences and occurrence to understand when to use which word correctly. This was the famous Natural Language Processing technique called Bag of words [1]. This predictive statistical sentence prediction lasted for a while until in 2014 a paper came for Neural Machine Translation NMT called sequence-to-sequence learning with neural networks [2], which was used by google to develop the conversational chatbots. This led to the basic understanding that if you take a sentence and encode it then pass it in a neural network that is acting like a decoder, then it can produce an output sentence with similar characteristics as the encoded data that was initially passed in, hence a conversational AI at its base level. Then in 2017 the current state of the art Transformer models got introduced in a paper "Attention is All You Need" [11] which brought about the current generation of language models. This was further enhanced in 2019 in development of the GPT-2 model [10] which can handle long sequences more effectively, making it suitable for a wide range of NLP tasks, including text generation, translation, and question answering.

II. Formal Problem And Evaluation Criteria

A. Formal Problem

This research project aims to build a conversation AI application that focuses on understanding the user conversation and reply to the user as a conversational friend with proper slang and manner matching a decently intelligent human conversation. The approach to build such an intelligent AI aims to use a Bidirectional LSTM Recurrent Neural Network Attention NMT model [1], and tweak its hyper parameters to achieve maximum accuracy and learnability, then further compare our model performance with the Transformer GPT-2 model and couple them together to get a output selection choice for different test cases. The project also aims to integrate live RASA actions API [4] features controlled by the conversational AI system, such that that it gives an updated response to dynamic questions like what is the current weather or time or the price for a given stock. The first major problem is zeroing down to the activation function, loss function and optimizer to be used in this sequence-to-sequence model for development. The second problem is the type of data to be used for training this model followed by the API architecture to link this AI with API services feeding in live data for conversational queries.

B. Evaluation criteria

The success of the research project is based on training the AI model to understand user conversation and output intelligent meaningful conversations that give a nice user chat experience. It should further understand detailed conversations asking for particular services, such as weather, time etc. and process such queries with live fetched service output. This AI will then be tested using BLEU score [3] (Bilingual Evaluation Understudy) that will be used to check accuracy with increasing sentence length, in order to evaluate how well the Conversation AI can process long complex conversational sentences.

III. Methodology

A. Steps to model training

This research project comprises of 4 steps – (1) Sourcing a Conversational dataset, (2) Developing a Conversational AI Chatbot model, (3) Rasa NLU Conversational Chatbot model coupling, (4) The API frontend for user experience. To understand these parts lets understand the steps to develop a Conversational AI –

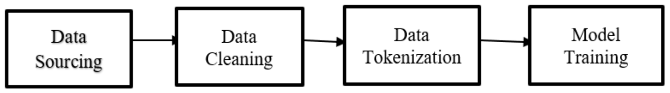


Fig 2 Data flow to a conversational AI

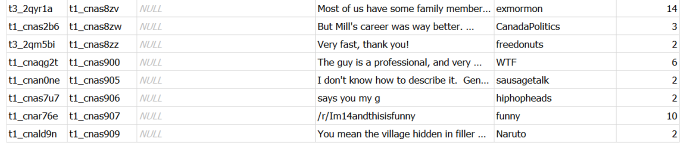
As Figure 2 shows, a Conversational AI development comprises of Data sourcing, Data cleaning, Data tokenization and Model training to develop a working chatbot using Machine learning. But the quality of a conversational chatbot is solely dependent on the type of data and models used to develop the chatbot. Let's understand the parts of this AI in detail.

B. Dataset used

This project used Reddit dataset comprising of 2.7 million conversations that took place on reddit. The data was sourced using the Reddit API for data up to year 2021. This provides the chatbot with a diversity of input data. As reddit conversations span multiple topics on current affairs, it makes the chatbot much more intelligent on how to use language. The diversity of the dataset also affects the usage of slang language and gives the chatbot a personality that replicates a human. For example, a regular chatbot replies with a generic tone like "Hi how are you" but this chatbot can even reply back with a "Hey! What's up?" because of reddit reply data. The dataset is broken down into a from.txt and to.txt target file for training purposes. These files are generated by traversing the data and scanning out all reddit answers with a score of 2 or more for asked questions. This score of minimum 2 is taken for answer validation purposes as any score less than that is not a correct answer, more over if an answer with a higher score than current score is found, then it replaces the current answer with the higher scored answer, thereby always fetching the highest scored answer for a question.

C. Dataset cleaning and tokenization

Once the relevant data is fetched it is cleaned for any blacklisted words and foul language found in reddit data, it is also cleaned of special characters that symbolize newline characters etc. This data is then saved in a MySQL database with the following schema structure - main_id TEXT PRIMARY KEY, comment_id TEXT UNIQUE, main TEXT, comment TEXT, redditthread TEXT, likes_score INT, the table is shown below.



main_id	comment_id	main	comment	redditthread	likes_score
t3_2qyr1a	t1_cnas8zv	NULL	Most of us have some family member...	exmormon	14
t1_cnas2b6	t1_cnas8zw	NULL	But Mill's career was way better. ...	CanadaPolitics	3
t3_2qm5bi	t1_cnas8zz	NULL	Very fast, thank you!	freedonuts	2
t1_cnaqg2t	t1_cnas900	NULL	The guy is a professional, and very ...	WTF	6
t1_cnan0ne	t1_cnas905	NULL	I don't know how to describe it. Gen...	sausagetalk	2
t1_cnas7u7	t1_cnas906	NULL	says you my g	hiphopheads	2
t1_cnar76e	t1_cnas907	NULL	/r/Im14andthisisfunny	funny	10
t1_cnald9n	t1_cnas909	NULL	You mean the village hidden in filler ...	Naruto	2

Fig 3 Database structure

Then this data goes through Byte pair encoding, which is a compression algorithm that is used in most modern NLP models like BERT. BPE algorithm is shown below [5].

```
function Byte-Pair Encoding (strings Str, number of merges z) returns vocab Vab
Vab ⟵ all unique characters in Str    # the tokens are characters
for i = 1 to z do                     #merge the token for z times
    tL, tR ⟵ Most frequent pair of adjacent tokens in Str
    tNEW ⟵ tL + tR                    # make new token by concatenating
    Vab ⟵ Vab + tNEW                  # Update the current vocabulary
Replace each occurrence of tL, tR in Str with tNEW   # and update the corpus data
return Vab
```

Byte Pair encoding has two major parts [5]. The first BPE step is a token learner. When raw text is sent in a token learner it generates a Bag of words, a unique set of words generated using all the input text, containing the frequency of the word recorded with the word itself in a key value pair manner. The token learner then generates a vocabulary by pairing two sets of characters that occur together with the highest frequency in the bag of words, the vocabulary keeps on increasing and pairs are formed using most frequent pair of words from the bag of words until all pairs are formed and vocabulary is generated. The second BPE step is a token segmenter [5], when raw sentences are passed in the token segmenter it tokenizes the most frequent words using the learned vocabulary. This leads to data tokenization of frequent words and also sub words like “est” or “un” called morpheme, where a morpheme is the smallest meaning bearing unit of a language.

D. Attention Models, Recurrent Neural Networks

To understand why a type of recurrent neural network is used to train this model we first understand why sequence to sequence models are used. A sequence-to-sequence model [2] is generally used for language translation. To learn a language, it is essential that the model learning it has a “memory” unit inside it, so that it can remember the previous sequence to predict the current sequence of words. For example, a sentence like – “United States is a democratic nation” has a sequence and a learning AI agent needs to remember the country name United States to predict the word “Democratic” as a finishing sequence of the sentence that describes what the word “United States” is by nature. That is why recurrent neural networks are used in a Sequence-to-Sequence model and it is called an attention model because as it has memory for the past sentence, and it pays “attention” to one word of a sentence at a time, hence earning the name “Attention Model”. Here the sequence-to-sequence model uses an encoder and a decoder [1]. Lets understand 2 major approaches in this field.

E. Transformer Models and NMT Models

The Transformer model and Neural Machine Translation (NMT) model are both significant advancements in the field of natural language processing (NLP) and have been widely used for various sequence-to-sequence tasks. Both models employ the attention mechanism, which allows them to effectively capture long-range dependencies in input sequences and improve their understanding of complex linguistic structures. In the Transformer model, self-attention layers are stacked to weigh the importance of different tokens in the input sequence when generating an output, making it highly suitable for tasks like machine translation and language generation. On the other hand, the NMT model specifically focuses on translating language sequences and utilizes an attention mechanism to align and translate the input and output sequences efficiently. The neural net design of both models involves multiple attention layers and feedforward neural network layers, allowing them to process large amounts of text data and achieve impressive performance in various NLP tasks. In comparison, the Transformer model is a more general architecture that can be applied to a wide range of NLP tasks, while the NMT model is specialized for translation tasks. Both models have demonstrated state-of-the-art performance and have paved the way for the development of more advanced language models like GPT-2 [10]. The neural net design of the Transformer model consists of an encoder-decoder. In the encoder, the input sequence is transformed into a series of hidden representations through self-attention layers. These layers calculate attention scores for each word in the sequence based on its relationship with other words in the same sequence, allowing the model to understand the context and dependencies between different words. The decoder then takes the encoded sequence and generates the output sequence by attending to the relevant parts of the input sequence using the attention mechanism. The model uses self-attention to ensure that each word in the output sequence is generated with respect to the relevant parts of the input sequence, leading to more accurate and contextually relevant translations. The neural net design of the Transformer model allows it to process sequences in parallel, making it significantly faster and more scalable than traditional recurrent neural networks (RNNs). The NMT model also uses a encoder and decoder architecture. Lets understand this in detail -

F. Encoder and Decoder

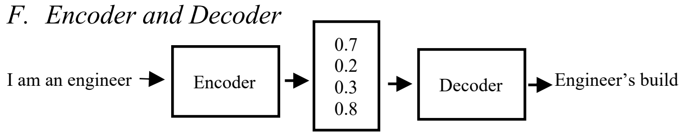


Fig 4 Encoder-decoder architecture

As shown above in Fig 4, an encoder takes in a tokenized word vector, builds a word annotation vector and passes it to the decoder so that the decoder can output the target words predicted for the input words. This is a basic NMT depiction. Let’s break down the details for the two. Firstly, the encoder and the decoder are two separate RNN’s [2]. The encoder is a Bidirectional RNN that primarily takes in the input data and condense its essential meaning in a condensed vector inside its hidden layer. This vector is then passed to the decoder at <EOS> end of sentence.

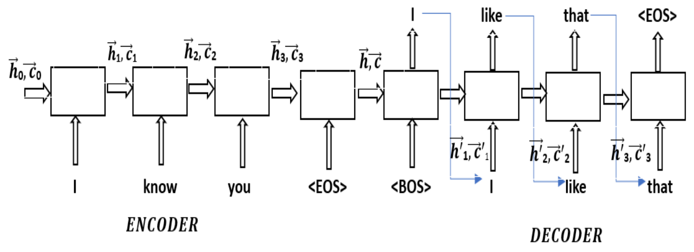


Fig 5 Encoder-decoder in-depth diagram

As shown the encoder neurons do not have an output prediction, but rather pass their current hidden state vector $\vec{h}$ and context vector $\vec{c}$ to the next word neuron, this continues until an entire $\vec{h},\vec{c}$ context vector is formed as shown in above image. Here as shown $\vec{h}_0,\vec{c}_0$, $\vec{h}_1,\vec{c}_1$, $\vec{h}_2,\vec{c}_2$, $\vec{h}_3,\vec{c}_3$, form the $\vec{h},\vec{c}$ vector where the input words are "I", "know", "you". Here the $\vec{h},\vec{c}$ vector is the encoder output, which is fed to the decoder as <BOS> beginning of sentence vector, that outputs prediction words using the previous fed context vector at each time step using a combination of Sigmoid and SoftMax activation function over the Decoder RNN. Here Sigmoid activation function controls flow of data inside the LSTM RNN memory cell. On the other hand, the SoftMax activation function is used to generate the output probabilities for the predicted word at each time step in the decoder LSTM. Here the output vectors $\vec{h'}_1,\vec{c'}_1$, $\vec{h'}_2,\vec{c'}_2$, $\vec{h'}_3,\vec{c'}_3$ represent the output words "I", "like", "that" until <EOS> end of sentence is reached. This is how an encoder and decoder work.

G. *Mini Batch hyperparameters selection*

The model is trained using mini batches to have balance between training speed and updating the model's weight more efficiently. In this model, the mini batch size is taken as 128. The encoder requires a 3D tensor as input. To make this 3D matrix, we first pass the mini batch sized sample data through a tokenizer which tokenizes or converts the sample of words to integer sequences using the vocabulary, here the vocabulary size is set to 30,000 words. Then a padding token <PAD> is added in front of the sequence to form a fixed sequence length size token, basically a 2D matrix of dimensions (mini batch size, sequence length), here the fixed max sequence length is set to 50 characters in this model. As a fixed size matrix is passed through the encoder therefore, we need fixed sized tokens. Then these tokens are passed through an embedding layer. This layer converts the tokens into continuous word vectors by adding a third embedding dimension to the 2D matrix above forming a 3D Tensor matrix with (mini batch size, sequence length, embedding dimensions) as dimensions. Here the embedding dimension is set to 512. Therefore, the input matrix 3D dimensions are (128, 50, 512). Then decoder takes in a similar matrix of dimension (mini batch size, sequence length, hidden dimensions), here hidden dimensions are set to 512 which is the same as embedding dimensions. This is how mini batches of training data are input to the encoder and decoder. Going forward we understand the Vanishing gradient problem.

H. *Vanishing gradient problem*

The vanishing gradient problem [6] occurs in recurrent neural networks specifically during the backpropagation. In this the gradient becomes so small that it becomes negligible, and weights stop getting updated, which prevents us from reaching the global minima.

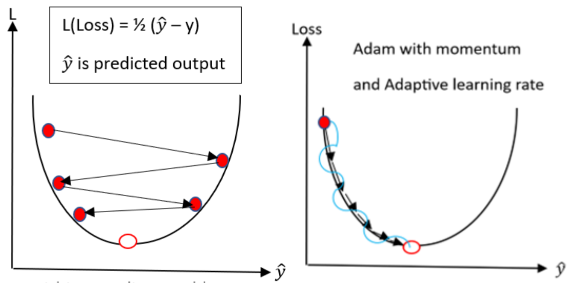


Fig 6 *Left*: gradient problem posed by sigmoid activation function, mitigated with LSTMs. *Right*: Speed and smoothness of convergence, mitigated with Adam Optimizer's momentum and adaptive learning rate [7].

As shown above here the gradient stops getting updated because of very small change in the activation value. This occurs due to the usage of Sigmoid function, as Sigmoid outputs a value between 0 and 1, the value can end up being too small or too large during the time-based backpropagation in recurrent neural networks. This happens because RNNs learn long patterns and keep memory of the past context vector and hidden layer vector state to give the current prediction. For long sentences, this tracking of past state of the network makes the current state significance small in comparison to the past total, making the activation output very small nearly negligible changes to output, thereby making the loss function give constant inaccurate loss and no update to weights for further training. To avoid the vanishing gradient problem RNN's use a gating mechanism for flow of data by using a LSTM RNN architecture. Going ahead we will understand how LSTM RNN work.

I. *Long Short Term memory RNN*

Below we see a diagrammatic representation of the LSTM memory gates.

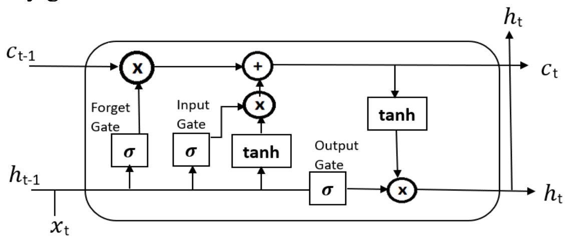


Fig 7 Long Short-Term Memory cell

An LSTM RNN [8] has a memory cell and this cell has an Input gate, Forget gate, and Output Gate. All these gates are used to regulate the flow of data inside the neural network. Therefore, the memory cell can hold on to long sentences of data. It is very important for a RNN to have this control as many time the prediction of a current word is often dependent on the previous sentence and remembering that previous sentence can help in prediction of current word. All these 3 gates are controlled by a sigmoid activation function which outputs a value between 0 and 1, which dictates when to open or close a gate. The forget gate is used to determine how much information has to be retained inside the network which is needed for upcoming prediction. The input gate takes in new data and adds it to the memory cell and the output gate determines how much current memory cell data should be given out as the memory cell output. In this way the gating mechanism is able to solve the vanishing gradient problem as it is able to selectively propagate gradient through time and prevent it from becoming too small. Here tanh is the hyperbolic tangent function which plays a vital role in managing and updating the cell state. The tanh function is a continuous, differentiable activation function that maps input values to a range between -1 and 1. It is defined as: tanh(x) = (e^(2x) - 1) / (e^(2x) + 1). The tanh function is applied to the LSTM for input modulation and output modulation. That is when the input decides the data that will be allowed to enter the memory cell, the tanh function is applied to the memory cell state to modulate the input. This is done by applying element wise multiplication between the input gates activation and the tanh activated memory cell state. This way the tanh function ensures that the memory cell state values are between -1 and 1, thereby providing a normalized and controlled update. After that the output gate specifies which part of the memory cell state will be exposed to the next time step. Before this data goes ahead, a tanh function is applied to the memory cell state, which normalizes the values between -1 to 1. Consecutively the output gate activation is applied element-wise to the tanh activated memory cell state, which gives the final output of the LSTM unit in RNN. But as the sentence length increases the accuracy of the model decreases. This happens because with increasing sentence length the LSTM memory cannot retain very long sentence data and consecutively the forget gate removes some context information which was required leading to decrease in accuracy with an increase in sentence length. To remedy this issue Bidirectional LSTM RNN is used let us understand the details.

J. Bidirectional LSTM RNN

The model used to train a conversational AI is a Seq2Seq Natural Machine Translation Attention model, which is basically a Bidirectional LSTM RNN NMT model [9] offered by google [12]. This model is changed in this project as per research requirements. The below figure shows a Bidirectional RNN and how it works –

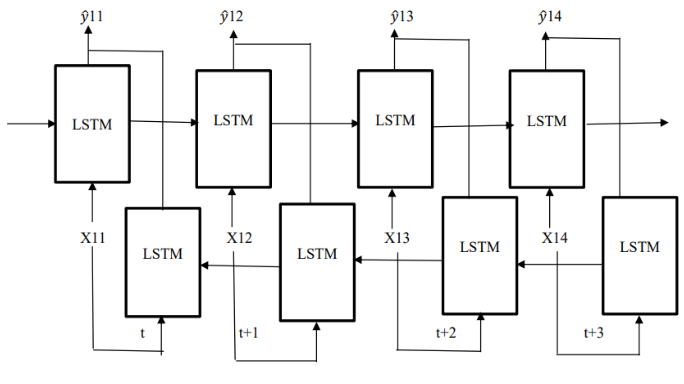


Fig 8 Bidirectional LSTM Recurrent Neural Network

As seen a Bidirectional LSTM RNN takes in an input vector <X11, X12, X13, X14> and outputs <yˆ11, yˆ12,yˆ13,yˆ14>. Here the BRNN connects two hidden layers of opposite directions to the same output. This is done by feeding the input vector first from the starting sequence X11, X12, X13, X14 and then in reverse from the end X14, X13, X12, X11 of the input vector. Both the stacked BRNNs run in opposite directions to each other and feed each other's predicted output to achieve a more accurate LSTM prediction. This is done to allow a long sentence sequence to get a correct sequence prediction. A regular LSTM RNN can forget long input context vector sequences, but BRNN architecture allows both future and past predictions to get used together to get a better current prediction output. This model is called an Attention model as like a translator it takes frames of a long sentence and translates these frames in parts individually to give a final output thereby paying attention to parts of a whole sentence one at a time. Here the encoder comprehends each word annotation of a given input sequence in the form of a hidden state vector, the encoder does this as it consists of BRNNs, it takes the sequence as an input and generates a final embedding at the end of the sequence. This is then passed into the decoder which produces the output sequence. Let us understand this in detail as shown in Figure 9 below. Here we have each output word conditional probability as p(yi|y1, . . . , yi−1, x) = g(yi−1, si, ci), over here si is the RNN hidden state and it is computed as si = f (si−1, yi−1, ci). As seen the context vector is ci which is used to get the probability of each target word yi. The context vector ci depends on a sequence of annotations (h_1, h_2,……h_{Tx}), and any annotation h_i has data on the entire input sequence with solid focus on the i-th word of the input sequence. Here we want the annotation of each word to summarize both the past and the future words. To achieve this a Bidirectional Recurrent neural network is used. An encoder is a BRNN that consists of feed forward and feed backward RNN's, the feed forward RNN reads an input sequence of inputs x1,x2,x3….xT and processes out a consecutive sequence of feed forward hidden layers ($\overrightarrow{h}_1$, $\overrightarrow{h}_2$, $\overrightarrow{h}_3$, $\overrightarrow{h}_4$,… $\overleftarrow{h}_{Tx}$).

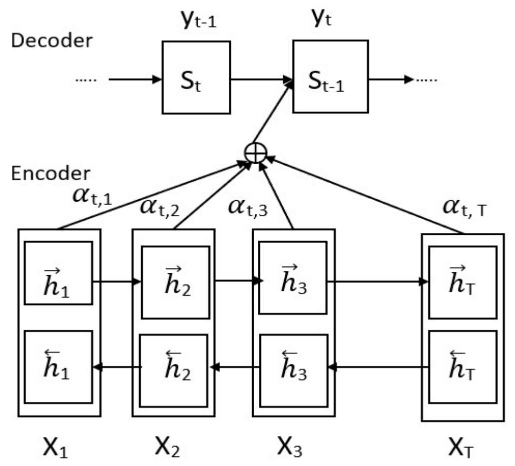


Fig 9 Bidirectional RNN encoder decoder architecture [9]

The feed backward RNN reads the sequence in reverse, that is like xT,….x3,x2x1 and it outputs feed backward hidden states ($\overleftarrow{h}_1$, $\overleftarrow{h}_2$, $\overleftarrow{h}_3$, $\overleftarrow{h}_4$,… $\overleftarrow{h}_{Tx}$). To get a combined annotation for each word x_r we concatenate both the feed forward and feed backward hidden states as $h_r = [\overrightarrow{h}_r^T; \overleftarrow{h}_r^T]$. This concatenation results in an annotation h_r which sum-ups the information for both the past and future words. As a RNN runs on a memory cell and focuses on recent words the annotation h_r provides it with information around the word x_r so that it can predict the current word better with more attention. This annotation h_r is passed to the decoder to create the context vector ci. The context vector ci is computed as weighted sum of all annotations h_i.

$C_i = \sum_{r=1}^{Tx} \alpha_i h_r$, here the weight α_{ir} is computed for each annotation h_r as seen below. $\alpha_{ir} = \frac{\exp(e_{ir})}{\sum_{k=1}^{Tx} \exp(e_{ik})}$, $e_{ir} = \alpha(s_{i-1}, h_r)$

This is basically an alignment model which scores the percentage match of inputs around position r and the output at position i. This score is set by the RNN hidden state s_{i-1} and the r-th annotation of input h_r. Here the probability of α_{ir} and its energy e_{ir} shows the significance of annotation h_r with respect to its previous hidden state s_{i-1}, this is used to decide its next state s_i and consecutively generating prediction probability y_i. This is used by the decoder to decide which part of the source sentence to pay attention to and this is how the attention model works [9]. This model uses Adam optimizer, the Cross Entropy loss function and Sigmoid activation function as hyperparameters.

K. *Inference model*

Once the model is trained, it needs to be used for conversations, this is what Inference models are used for. The inference model refers to the process of using a trained sequence-to-sequence model with attention to generate translations for new input sentences in the source language [1]. The primary goal during inference is to find the most likely output sequence that is a sentence in the target output specified given the input sequence. The inference model on input takes the same steps as stated before, it preprocesses input sequence using BPE to form tokens, pass them through an encoder to get the hidden states that represent the input data. This is then passed to the decoder, the decoder computes the attention score using the encoder's hidden state based on decoder's current state. It then calculates the context vector as a weighted sum of encoder's hidden states, using the attention score as weights. It combines the context vector and the decoder's hidden state to predict the next output token and updates the decoder's hidden state based on the predicted output token. Now to generate the output sequence there are many strategies. Like Greedy search or Beam Search [1], let us understand them in detail. In Greedy search at each decoding step, we select the token with the highest probability and feed it to the decoder as input for the next step. This approach is fast and yields a single output. The problem is that we can never be sure whether this single output does justice with the conversational patterns our Attention model is capable of producing and hence we use a technique called Beam Search to overcome this issue. In Beam search, we take a fixed number of partial output sequences called "Beams" at each step. We then expand each beam by considering all possible next tokens and their probabilities. We keep only the top beams according to their cumulative probabilities. By doing so we can accumulate the best output token for each step because we have more variety to choose from. This approach can find better translations than greedy search but is more computationally expensive. Here Beam width is a hyperparameter that decides the number of computations for each output Beam basis the number of partial output sequences it holds. Finally, when the output sequence is generated usually terminated by an <EOS> end of sequence token or maximum set length, we do the postprocessing in which we detokenize the output sequence, check for incorrect punctuation, capitalize starting words, check for any unprofessional language for removal and output the final sentence for the conversation.

IV. Results

A. *BLEU Scores*

In this research project Sequence to Sequence prediction quality is tested by using a BLEU score [3] (BiLingual Evaluation Understudy). The BLEU score is a measure of how closely the predicted output matches the reference answer. Below we see graphs showing BLEU score for a BRNN. The graph below is a projection of how our NMT model trained and became better at prediction.

(A) BLEU score up to 50000 Mini Batch iterations

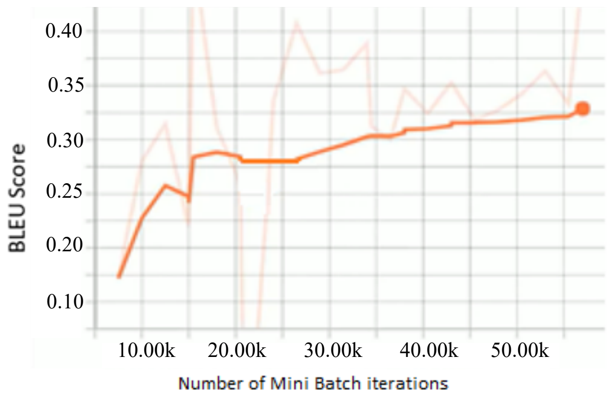


Fig 10 BLEU score progression with increasing steps

Here in Figure 10 (A) we see BLEU score against the number of mini batch iterations, we see as the model starts to learn and the BLEU score starts increasing after 10000 steps of training and we see its consistency after about 30000 steps of training. Consecutively with each increasing mini batch iterations, the model learns how to predict better output vocabulary tokens and the BLEU score keeps on increasing. The BLEU score depicts that even with increasing sentence length the model is able to correctly predict the output sequence.

(B) Test Set Results

In this section, we present our evaluation methodology using a test set comprising 5000 conversations of varying lengths. These conversations are used to generate output predictions using our trained Neural Machine Translation (NMT) model and are compared against the Transformer-based pretrained GPT-2 model, a well-known benchmark. To assess the quality of the generated responses, we calculate the BLEU score for each output prediction. The BLEU score is a measure of how closely the predicted output matches the reference answer. We average the BLEU scores calculated using the model's output for different input sentence lengths to check model's power of remembering sentence context, ranging from 1 to 50 input length in the case of the NMT model and 100+ input length for the Transformer GPT-2 model. Figure 11 illustrates the results of our evaluation. We observe that our trained NMT model performs well, achieving a good BLEU score for input sentence lengths of up to 40. However, as the input sentence length approaches the set threshold of 50, the prediction quality starts to decline, resulting in a decrease in the BLEU score. It is important to note that a BLEU score of 1 represents the maximum achievable score, indicating a perfect match between the predicted and actual reference answer.

BLEU Score

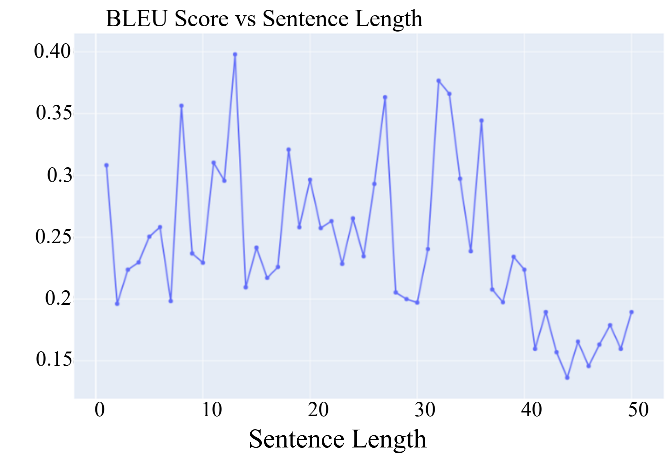


Fig 11 Test results for NMT Attention model

This evaluation provides valuable insights into the performance of our NMT model and its ability to generate accurate responses for varying output lengths. By comparing it to the well-established GPT-2 model, we gain a deeper understanding of the strengths and limitations of our approach. These findings contribute to the overall understanding of conversational AI systems and inform future advancements in the field.

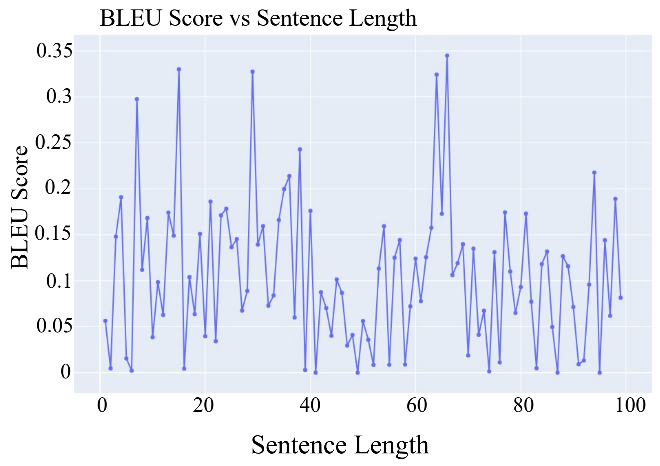


Fig 12 Test results for Transformer GPT-2 model

As seen above in figure 12 Transformer based GPT-2 model has a consistent performance for way more then 100 sentence length and is better in performance in comparison to the NMT model. Plus the GPT-2 model is easily couplable with the NMT model within the conversational AI application. Giving a choice of selection by developing an answer scorer and scoring individual outputs of the 2 models before returning a better scored answer in the output.

V. Live Service Architecture

A. Rasa Actions API

Just like an Attention model, RASA NLU [4] a famous chatbot development tool also uses Bidirectional LSTM RNNs to train chatbots. A new feature of live Actions API using Rasa is added to the project. This feature supports conversational queries like "what is the temperature in location name" or "what is todays stock price for company name". Such conversations require fetching data from live services before giving an answer and the conversational AI is well capable of understanding and processing such conversations, and provides live data using Rasa's Actions server. Let us understand this in detail.

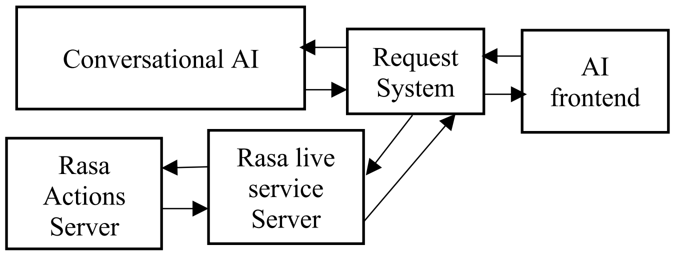
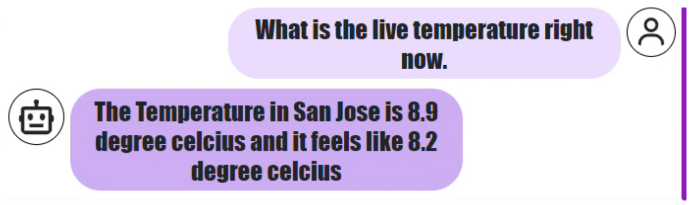


Fig 13 Rasa Live Actions Architecture system

Once a request is made from the frontend it sends a request to the request system which checks the nature of the request. If the request requires live data it is sent to the Rasa server. Which calls the Actions server. This gives us the capability to query live service requests using weather and stock API's and return live data for the conversation. We specifically use Rasa actions API for this as it gives a more story-based approach to live request conversations [4]. This is achieved by training a model in Rasa. To do so we first have set intents and their examples in the nlu.yml file, the examples are the example conversations which will be requested by the user for a particular scenario like asking the weather, time or stock price. Then we set the story pattern which is basically a script of what to expect in a conversational pattern. Like in a basic conversation, someone will say hi, ask your name, and a set of follow up questions. Such conversational patterns are set in the stories.yml file for Rasa. Then we write the domain.yml file where we map different intents to their utter responses. This file dictates a set of responses to an intended conversation. It also dictates actions to certain intended conversations and this is where we map an action to its actual process that occurs in the action.py file. The actions.py runs on a Rasa actions server. This is a standalone module separate from Rasa that allows Rasa to communicate to run python code in form of an action to perform different actions. Be it querying the database, or sending a request to a live service API like a weather forecasting API to ask for the current weather at a specified location. As during training, we set names of locations in the intents and domain response files, Rasa learns to pick up location data from the conversation by using different sample queries to understand the request nature and same goes for the stock market API company name querying or the time request API. In the below image we see how it works on the developed conversational AI.


What is the live temperature right now.

The Temperature in San Jose is 8.9 degree celcius and it feels like 8.2 degree celcius

Fig 14 Real time data chat response

As we can see in Fig 14 when asked for the live temperature, the conversational AI request RASA server for the information, which in turn queried the request to its actions server which requested the weather API, collected data and returned it as a reply to conversation.

B. *Micro-Service Architecture and Fault Tolerance*

This research project was developed on a microservice architecture. As the Conversational AI has a React frontend, the Flask API core manages two different models, one is the main NMT Chatbot model which can speak on any topic and the second is a Rasa model specifically coupled in for live data requests handling. The Conversation AI model being trained on 300 Gigabytes of data has a size of 3.5 Gigabytes itself and the pretrained GPT-2 model has size of 547 MB. The Rasa model is again 1.3 Gigabytes in size. All 3 models work together for giving a nice user chat experience. As this is a Conversational AI it is trained to give answers to all questions. A conversation scoring mechanism scores the quality of replies from the Conversational AI model, the GPT-2 model and the Rasa model and returns the best scoring answer as a response to a conversation. Below is an architectural diagram of the system and its parts –

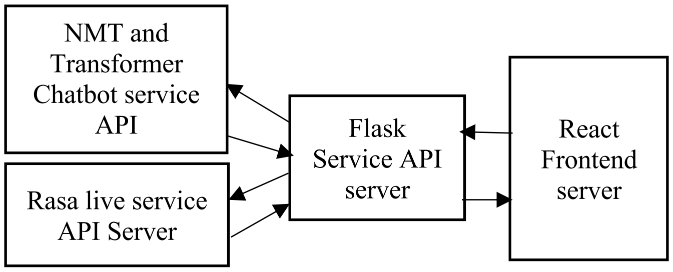

NMT and Transformer Chatbot service API
Rasa live service API Server
Flask Service API server
React Frontend server

Fig 15 Micro-Service Architecture of Conversational AI system

As shown above, the react server sends a JSON request to the flask API. The API then forwards the request to both NMT and Transformer chatbot server and Rasa server so both can participate in answering the conversation, otherwise if it is a complex question, it is sent solely to the NMT and Transformer Chatbot server and if it is a live data request it is sent solely to the Rasa Server. The nature of the conversation is determined at the Flask Service API level and the response is also given back to the react server from it. The system is fault tolerant because if the NMT and Transformer Chatbot cannot solve a query the Rasa server will be able to and vice versa.

VI. Conclusion

In conclusion, this research project has successfully addressed the limitations of traditional industrial chatbots by developing an innovative approach that enables chatbots to provide real-time and up-to-date information to users. By harnessing the power of Neural Machine Translation (NMT) and Transformer Models, we created a conversational AI agent that can deliver highly interactive and data-driven responses. The microservice architecture, coupled with the RASA actions API, has allowed us to efficiently integrate these models with live API services, making the chatbot flexible and cost-effective to develop. Our findings demonstrate that by optimizing hyperparameters and leveraging open-source technologies, even small or medium sized organizations can afford to deploy interactive AI chatbots that give a human feel in responses. This approach goes beyond traditional scripted responses, offering a personalized and dynamic user experience. With the ability to provide real-time information, the chatbot becomes a valuable tool for users seeking live updates and engaging interactions. The success of this research project paves the way for more accessible and powerful conversational AI technology, benefiting businesses of all sizes and users alike. As the field of NLP continues to advance, we anticipate that these innovations will lead to even more sophisticated chatbots with broader applications in various industries.

Acknowledgment

Research reported in this publication was supported by the Division of Research and Innovation at San Jose State University under **Award Number 23-SRA-08-028**. The content is solely the responsibility of the authors and does not necessarily represent the official views of San Jose State University.

References

[1] S. Yang, Y. Wang, and X. Chu, "A Survey of Deep Learning Techniques for Neural Machine Translation," Journal of Computer Science and Technology, Jul. 2020. doi: https://doi.org/10.48550/arXiv.2002.07526

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks", 2014. doi: https://doi.org/10.48550/arXiv.1409.3215

[3] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, "BLEU: a Method for Automatic Evaluation of Machine Translation," in Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002), Philadelphia, Pennsylvania, USA, 2002, doi: https://doi.org/10.3115/1073083.1073135

[4] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa: Open Source Language Understanding and Dialogue Management. doi: https://doi.org/10.48550/arXiv.1712.05181

[5] K. Bostrom and G. Durrett, "Byte Pair Encoding is Suboptimal for Language Model Pretraining. doi: https://doi.org/10.48550/arXiv.2004.03720

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157-166, 1994. doi: https://doi.org/10.1109/72.279181

[7] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization" doi: https://doi.org/10.48550/arXiv.1412.6980

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation Nov. 1997. doi: https://doi.org/10.1162/neco.1997.9.8.1735

[9] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate. doi: https://doi.org/10.48550/arXiv.1409.0473

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019. doi: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need", 2017 doi: https://doi.org/10.48550/arXiv.1706.03762

[12] M. T. Luong, E. Brevdo, and R. Zhao, "Neural Machine Translation (seq2seq) Tutorial," https://github.com/tensorflow/nmt. Accessed on April 12th, 2023.